非最短経路を用いたチップ内ネットワーク向け経路設定手法

松谷宏紀[†] 鯉渕道紘[††] 山田 裕[†]
上樂明也[†] 天野英晴[†]

本論文では，チップ内ネットワーク向けに *flee* と呼ばれる固定型ルーティング法を提案する．*flee* では非最短経路を導入することで，局所性の強いトラフィックをネットワーク全体に分散させ，スループットを向上させる．近年の Systems-on-a-Chip 設計では，アプリケーションは SystemC に代表されるシステムレベル言語で記述され，設計の初期段階からシミュレーションされる．この段階で各ノードのタスク割り当てが決まるので，ノード間の通信パターンを解析できる．*flee* では，この解析結果をもとに，多量のデータが流れるソース-ディスティネーション・ペアに優先的に最短経路を割り当てる．一方，データ転送量の少ないペアは高負荷なチャネルを避けるように経路が張られるため，非最短経路を取ることがある．実アプリケーションを用いた評価では，*flee* ルーティング法を用いることによって dimension-order ルーティングより最大 28.6%スループットが向上した．

Non-Minimal Routing Strategy for Networks-on-Chips

Hiroki Matsutani,[†] Michihiro Koibuchi,[††] Yutaka Yamada,[†]
Akiya Jouraku[†] and Hideharu Amano[†]

We propose a deterministic routing strategy called *flee* which introduces non-minimal paths in order to distribute traffic with a high degree of communication locality in Networks-on-a-Chip. In the recent design methodology, target system and its application of the Systems-on-a-Chip are designed in system level description language like SystemC, and simulated in the early stage of design. The task distribution is statically decided in this stage, and the amount of traffic between nodes can be analyzed. According to the analysis, a path that transfers a large amount of total data is firstly assigned with a relaxed limitation, thus it is mostly minimal. On the other hand, paths for small amount of total data, are secondly established so as not to disturb previously established paths, thus they are sometimes non-minimal. Simulation results show that the *flee* routing strategy improves up to 28.6% of throughput against the dimension-order routing on typical stream processing application programs.

1. はじめに

Systems-on-a-Chip（SoC）において，IP コア同士を結合するチップ内結合網は，アプリケーションの性能とコストを決定付ける一要素である．チップ内結合網として従来から用いられているバスは，様々な性能改善手法が提案されているが，バスにつながるノード数が増えるとバスが通信のボトルネックとなる傾向がある．また，近年の集積技術では，ゲート遅延よりも配線遅延の影響が深刻であり，長い配線を構成するバス構造は動作周波数においても不利である．さらに，近年，多数の演算要素や簡単なプロセッサをタイル状に接続したリコンフィギャラブルシステムやオンチップマルチプロセッサが登場し[1),2)]，多数のコアを接続するためのチップ内ネットワーク（Networks-on-a-Chip）[3)~8)]の研究が盛んになっている．

チップ内ネットワークでは，従来，並列計算機や System Area Network（SAN）で用いられてきたパケット転送技術をチップ内の IP コア間のデータ転送に応用する．これらのネットワークでデータを送信する場合，送信元ノード☆がデータにヘッダを加えパケット化し，中継ノードがこれを転送，最終的に宛先ノードでデータが取り出される．複数のパケットが同時に同一チャネルを取り合わない限り，複数パケットを並列に転送できるので，バスよりも通信帯域に優れる．チップ内ネットワークでは，パケットを構成する各フリットは固定長のリンク上を移動し，中継ノードごとにバッファリングされる．そのため，グローバル配線はいくつかの隣接ノード間配線に置き換えられ，配線遅延の問題も解決できる．さらに，チップ内ネットワークにエラー訂正や再送技術を導入することで，さらなるプロセス微細化よって今後深刻となるクロストークの問題を解決できると期待されている．

特定用途向けに製造されるチップでは，各ノードは小規模な構成で実装されると考えられる．そのため，各ノードが持つルータも小規模なほうが好ましく，文献 4)~8) では単純な固定型ルーティングを採用している．ネットワーク構造には，本来，様々なアプリケーションに適用するための柔軟性やスケーラビリティが求められる．しかし，SoC

† 慶應義塾大学大学院 理工学研究科
Graduate School of Science and Technology, Keio University
†† 国立情報学研究所
National Institute of Informatics

☆ 本論文では，チップ内ネットワークによって結合される IP コアをノードと呼ぶ．

では一度製造された後にノード数，ノードの性能や機能，ネットワークトポロジが変化することは考えにくい．そのため，チップ内ネットワークにおいては，汎用性を重視するよりも個々のアプリケーションに特化したネットワークアーキテクチャのほうが適している．

SoC の主要なアプリケーションは組込み機器であり，メディア処理や通信分野では高負荷なストリーム処理[9] が行われることも多い．近年の SoC 設計では，対象アプリケーションは SystemC などのシステムレベル言語で記述され，設計の初期段階からシミュレーションされる．この段階で，各ノードのタスク割り当てが決まるのでノード間のデータ転送量をある程度見積もることができる．多数のタイル構造を持つリコンフィギャラブルシステムやオンチップマルチプロセッサにおいても，対象アプリケーションとマッピングが決まれば，同様にノード間の転送量の見積もりが可能である．文献 10) では見積もった通信パターンを解析し，アプリケーションに適したトポロジを生成する設計手法を提案している．しかし，チップ内ネットワークにおける経路設定では，通信パターンを考慮しない並列計算機向けの最短経路ルーティング[11] が用いられているのが現状である．

本論文では，*flee* と呼ばれるチップ内ネットワーク向け固定型ルーティング法を提案する．*flee* では SoC の設計段階で得られるデータ転送量の見積もりをもとに，非最短経路を含めた経路設定を行う．ストリーム処理ではアクセスが一ヶ所に集中するなど，通信パターンに強い局所性が出やすい．この場合，トラフィックが同一チャネルに集中しないように経路を分散させることが望ましいが，最短経路だけでは経路長が短く十分にトラフィックを分散できるとは限らない．そこで，*flee* ルーティング法では，非最短経路を導入することで代替経路の候補数を増やし，高負荷なチャネルを確実に回避する．その際，通信パターンを解析し，多量のデータが流れるソース-ディスティネーション・ペアにできるだけ最短経路を割り当て，データ転送量の少ないペアに非最短経路を割り当てる．

本論文の構成は次の通りである．まず，2 章で SoC の主要なアプリケーションであるストリーム処理の特徴を述べる．3 章でチップ内ネットワークで利用可能な既存のデッドロックフリーなルーティング法を説明し，4 章で *flee* ルーティング法を提案する．そして，5 章で実アプリケーションを用いた *flee* の評価結果を示し，6 章で本論文をまとめる．

2. ストリーム処理

Viterbi デコーダや JPEG，MPEG コーデックのようなストリーム処理では，ひとかたまりのデータに対し一連の処理が実行される．本論文では，このような一連の処理の流れのうち 1 つの処理単位をタスクと呼ぶ．**図 1** に JPEG2000 デコーダのタスクの流れを示す．各タスクはそれぞれ各ノードに割り当てられ，パイプライン的に動作する．図 1 ではタスク間通信は隣接ノード間に限られている．この JPEG2000 の例では，EBCOT（Embedded Block Coding with Optimal Truncation）の計算負荷が

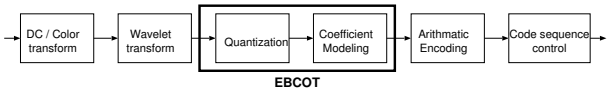

図 1 JPEG2000 デコーダのタスクの流れ．EBCOT の負荷が高い．

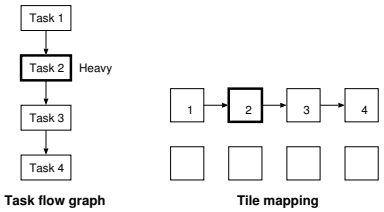

図 2 逐次実行モデル．タスク 2 が高負荷．

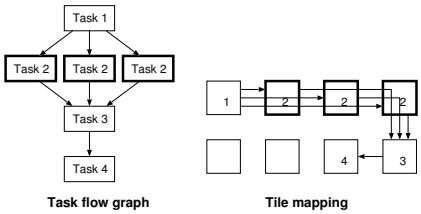

図 3 並列実行モデル．タスク 2 は 3 つのノードに分散され並列処理される．

高く，これらをそれぞれ 1 つの計算ノードで実行すると EBCOT が処理のボトルネックとなる（**図 2**）．この場合，**図 3** に示すように，EBCOT を複数のノードに分散し並列に実行することで，ストリーム処理の負荷を均等にできる．図 3 の並列化されたモデルでは，ノード間通信に分散と収集が発生する．

小規模な SoC において，メモリ，I/O，プロセッサなど IP コアの特徴と種類が明確に決まっていれば，ネットワーク自体を用途に合わせて作り込むこともできる．このような場合，ネットワークの経路設定に自由度はほとんど必要ない．しかし，本研究では，接続する IP コア数が多いオンチップマルチプロセッサやリコンフィギャラブルシステムなど大規模な SoC を対象とする．これらのチップ内ネットワークでは，タイル構造に対応した 2 次元メッシュ[1),2),4),6)~8)] やトーラス[3),5)]，Fat ツリーなどのネットワークトポロジが利用されることが多い．タスクをこのようなトポロジに割り当てると，図 3 に示すようにノード間の通信に高い局所性が生じる．このような hot-spot はパケットの衝突を引き起こし，スループット低下の原因となる．したがって，高い局所性が生じる箇所を避けるかたちでトラフィックを分散させれば性能を改善できる．

3. 既存のルーティング法

SAN やチップ内ネットワークで利用されるルーティング法は，適応型ルーティングと固定型ルーティングに大別できる．適応型ルーティングではパケットごとに動的に通信経路が選択されるが，固定型ルーティングでは静的に経路が定まる．固定型ルーティングの利点は，出力チャネルを動的に選択する機能が不要なことと，パケット転送の FIFO 性が保証されることである．文献 5) で示されるように，チップ内ルータのハードウェア量は無視できない．コスト制限

が厳しいチップ内ネットワークでは，より単純な固定型ルーティングが適している．

チップ内ネットワークで広範囲に利用されている固定型ルーティングとして，dimension-order ルーティング[11] が挙げられる．dimension-order ルーティングでは，2 次元メッシュやトーラスにおいて，まず，送信元ノードから x 軸方向のチャネルを使って移動し，次に，y 軸方向のチャネルを使って宛先ノードに到達する．dimension-order ルーティングは文献 5)~8) で利用されている．

一方，適応型ルーティングによって提供された経路集合の中から 1 つの経路をあらかじめ選択し，静的に経路を設定することで固定型ルーティングを実現できる．文献 12) では，各リンク上を通過するソース-ディスティネーション・ペアの数に注目し，一ヶ所にパスが集中しないように経路を選択する．本論文ではこの経路選択アルゴリズムを Sancho のアルゴリズムと表記する．一方，単純な経路選択アルゴリズムとして，ランダムに経路を選択する方法（random），小さい番号の出力ポートの経路を選択する方法（lowport-first）がある[13]．これらの経路選択アルゴリズムの性能比較[13] より，Sancho のアルゴリズムを使えば安定的に性能を向上できることがわかっている．

様々な並列プログラムが動作する SAN では，発生するトラフィックのデータ転送量を予測することは難しい．既存の経路選択アルゴリズム[12)13)] はこのような SAN 向けに設計されているため，各ソース-ディスティネーション・ペアのデータ転送量に応じて経路を選択する機能がない．しかし，図 3 で示したように分散と収集を含む通信パターンでは，ソース-ディスティネーション・ペアごとにデータ転送量が大きく異なる．そして，パイプライン処理のメインストリーム同士が同一チャネルを取り合うようでは性能は出ない．よって，データ転送量に応じて経路を構築する手法を提案することで，既存のルーティング技術に比べチップ内ネットワークのスループットを向上できると考えられる．

4. *flee* ルーティング法

本章では，非最短経路を活用する固定型ルーティングとして *flee* ルーティング法を提案する．2 章で述べたとおり，ストリーム処理の通信パターンには強い局所性が生じやすい．そこで，*flee* ルーティング法では，非最短経路を導入することで代替経路集合の数を増やし，経路を分散させて混雑を緩和する．その際，各ソース-ディスティネーション・ペアのデータ転送量を考慮して経路を設定する．具体的には，多量のデータが流れるソース-ディスティネーション・ペアには優先的に最短経路を割り当て，データ転送量の少ないペアは高負荷なチャネルを避けるような経路を設定する．

flee ルーティング法は次の 2 ステップから構成される．(1) 通信パターンの静的な解析，(2) デッドロックフリーな条件のもとでの経路設定．

4.1 通信パターンの解析

近年の SoC 設計では，対象アプリケーションは SystemC

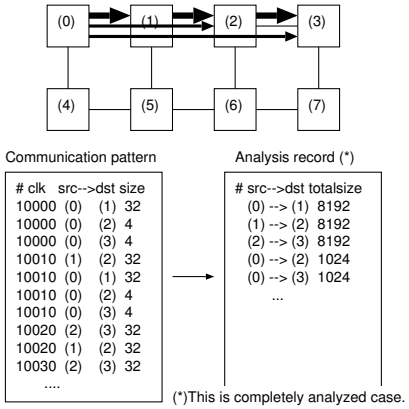

図 4 通信パターンの解析．データ転送量の多い順にソース-ディスティネーション・ペアをソートする．

や SpecC などのシステムレベル言語で記述され，設計の初期段階からシミュレーションされる．この段階でノードへのタスク割り当てが決まり，SystemC レベルのシミュレーションによって通信パターンを解析できる．タイル構造を持つリコンフィギャラブルシステムやオンチップマルチプロセッサにおいても，対象アプリケーションとマッピングが決まれば，通信パターンの解析は可能である．この通信パターンの解析結果をもとに，以下の手順で各ソース-ディスティネーション・ペアに優先順位を付ける．

(1) ソース-ディスティネーション・ペアごとに，対象アプリケーションで発生するデータ転送量の合計を求める．

(2) データ転送量の合計値が多い順に，ソース-ディスティネーション・ペアを並び換える．

図 4 を用いて通信パターンの解析手順を説明する．まず，SoC 設計のシミュレーション結果をもとに，パケット発生時のクロック，送信元ノード，宛先ノード，データサイズから成る通信パターンを得る．そして，データ転送量の多いソース-ディスティネーション・ペアから順に高い優先順位を与え，これを “analysis record” として記録する．次のステップでは，データ転送量が多く高い優先順位を得たソース-ディスティネーション・ペアから優先的に最短経路が割り当てられる．

Viterbi デコーダや JPEG，MPEG コーデックのようなストリーム処理では，通信パターンの静的解析が実現できる[9)] ので，タスク間通信のデータ転送量の見積もりが可能である．しかし，ストリーム処理以外のアプリケーションでは，データフローが動的に決まるなど，経路設定段階に及んでもタスク間通信のデータ転送量が見積もれないことがある．つまり，図 4 の通信パターンで送信元ノードと宛先ノードの集合のみが限定されている場合である．データ転送量を含んだ analysis record のほうが経路を分散させるためには有利であるが，*flee* はデータ転送量が不明な場合にも適用できる．このような不完全な analysis record と完全な analysis record を用いた *flee* の性能比較は 5.2.3 節で示す．

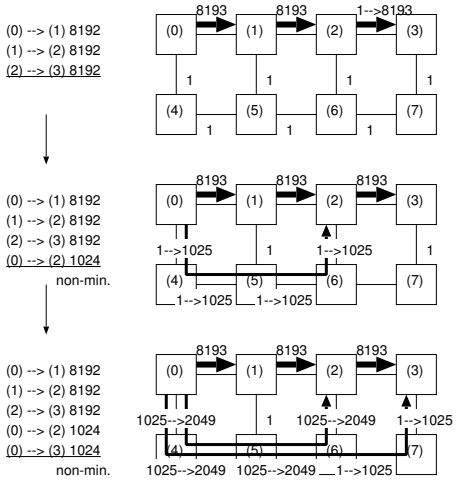

図 5 *flee* における経路設定．(0)-(2) 間と (0)-(3) 間の通信に非最短経路が割り当てられている．

4.2 経路設定

前節で説明した analysis record の優先順位をもとに，各ソース-ディスティネーション・ペアにデッドロックフリーな経路を割り当てる．ここで，ネットワーク内の全てのチャネルにコストという指標を導入する．代替経路が複数ある場合，それぞれの経路で利用されるチャネルのコストが比較され，コストが最も小さい経路が採用される．経路設定手順を次に示す．

(1) 全てのチャネルのコストを 1（minimum channel cost）に初期化する．

(2) 経路が設定されていないソース-ディスティネーション・ペアの中から一番優先順位が高いものを 1 つ選ぶ．そのペアに対して：

  (a) Dijkstra 法☆を用いて，デッドロックフリーの条件を満たす最小コストの経路（必ずしも最短経路とは限らない）を選択する．

  (b) 選択した経路が通過する全てのチャネルのコストを増加させる．増分は，そのペアによるデータ転送量の合計値，または，1 とする．

(3) すべてのソース-ディスティネーション・ペアの経路が定まるまでステップ 2-3 を繰り返す．

ステップ 2 (a) のデッドロックフリー保証は，Turn-Model[14] など適応型ルーティング等で用いられるデッドロックフリー条件を課すことで実現する．ステップ 2 (b) でソース-ディスティネーション・ペアが完全に解析できた（データ転送量が判明している）場合，各チャネルコストにデータ転送量の合計値を加算する．一方，完全に解析できなかった場合，データ転送量が不明なので単純に 1 を加算する．

データ転送量の合計値の単位として，ビット長やパケット数，パケット長を用いることができる．ただし，SoC 設計の初期段階におけるアルゴリズムレベルのシミュレーションでは，ネットワークの詳細設計が済んでおらず，パケット数やパケット長の情報は利用できない可能性がある．

☆ 重み付き有向グラフにおいて最短経路を発見するアルゴリズム．

図 5 に *flee* による経路設定の例を示す．この例は，ソース-ディスティネーション・ペアのデータ転送量が完全に解析できた場合を想定しており，デッドロックフリー条件として West-first Turn-Model[14] を採用した．図 5 に示すように，優先順位の高いソース-ディスティネーション・ペアは最初に経路が設定されるため，結果的に最短経路が割り当てられる．経路設定が進むにつれて各チャネルコストは初期値の 1 から増加し，hot-spot となるチャネルではコストが高くなる．優先順位の低いペアでは，hot-spot を避けるように経路が張られるため，非最短経路を取ることがある．このように *flee* ルーティング法では，通信のメインストリームに最短経路を割り当てる一方，優先順位の低い通信には，ネットワークリソースを有効活用するために非最短経路を割り当てることがある．

flee ルーティング法の計算量は $O(n^2k)$ である．ただし，n はネットワーク中のノード数，k は経路を設定するソース-ディスティネーション・ペアの数とする．

5. 性能評価

flee ルーティング法は複数経路を持ついかなるトポロジにも適用できるが，ここでは，タイル構造に適合し，既存のルーティング法の研究が進んでいる 2 次元メッシュおよびトーラス上で *flee* の評価を行う．さらに，一部のリンクが欠損したメッシュを想定し，up*/down*ルール上に *flee* ルーティング法を適用，*flee* と既存の経路選択アルゴリズム[12] を比較する．メモリの製造プロセスでは，冗長メモリセルを用いた故障箇所の修復[15] によって歩留まりを向上させている．SoC においても，今後，このような耐故障性技術が利用される可能性があるため，本論文ではリンク欠損を含んだメッシュでの評価も行う．

多くのストリームアプリケーションでは設計時に通信量を解析できるので，上記の評価では完全に解析された analysis record を利用する．一方，本章の最後の節では，完全な analysis record と不完全な analysis record を用いた *flee* の性能比較を行う．

5.1 シミュレーション環境

5.1.1 ネットワークモデル

評価のために C++言語で記述されたフリットレベル・シミュレータを実装した．評価対象のトポロジとして，チップ内ネットワークで一般的な 2 次元メッシュ[4),6)~8)] とトーラス[3),5)] を用いる．各ルータは 5 つのポートを持ち，1 つはノードとルータの接続に，残りは隣接ルータとの接続に用いる．各ルータのスイッチング機構には，チャネルバッファ，クロスバ，リンクコントローラ，コントロール回路を単純化したモデルを採用した．ヘッダフリットの転送には 3 サイクルかかる．具体的には，ルーティングに 1 サイクル，フリットがクロスバを通り入力チャネルから出力チャネルに到達するのに 1 サイクル，フリットが次のルータまたはホストに到達するのに 1 サイクルである．シミュレーション時間は 1,000,000 サイクル以上とした．

トレースデータの時間軸を変化させることで，様々な注

入データレートに対するスループットとレイテンシの評価を行う．これは，ノード内の処理時間をネットワークに対して相対的に変化させた場合の評価にあたる．ノード内の処理時間には，アプリケーションの計算時間の他に，パケットの生成，解体によるオーバヘッドも含まれる．

リンク欠損のないメッシュとトーラスでは，*flee* ルーティング法のデッドロックフリー制約条件として West-First Turn-Model[14] を適用する．この場合，*flee* および dimension-order ルーティングが利用する仮想チャネルの数はメッシュで 1，トーラスで 2 である．一方，リンク欠損のあるメッシュでは up*/down*ルールを適用する．この場合，*flee* と他の経路選択アルゴリズムの仮想チャネル数は 1 となる．

リンク欠損のあるメッシュでは，リンク欠損なし，2 リンク欠損の場合についてそれぞれシミュレーションする．欠損リンクは，各シミュレーションごとに最も負荷の高いリンクの中からランダムに選択する．up*/down*ルールには，同一の幅優先スパニングツリーを用いる．up*/down* ルールにおける *flee* ルーティング法は，次の 2 つの経路選択アルゴリズムと比較される．(1) ランダムな経路選択（random），(2) トラフィックの静的解析をもとにした SAN 向け経路バランシングアルゴリズム（Sancho のアルゴリズム）[12]．Sancho のアルゴリズムでは，各リンク上を通過するソース-ディスティネーション・ペアの数に注目し，一ヶ所にパスが集中しないように経路を選択する．Sancho のアルゴリズムでは各ソース-ディスティネーション・ペアのデータ転送量は考慮しない．

5.1.2 トラフィックパターン

flee ルーティング法では通信パターンをもとに経路を設定するため，実アプリケーションの通信パターンを用いて評価することが望ましい．そこで，典型的なストリーム処理アプリケーションとして JPEG コーデック，Viterbi デコーダ，IPsec アクセラレータのトレースデータを評価に用いる．これらの実装は，NEC エレクトロニクス製の動的リコンフィギャラブルプロセッサ DRP[16] 向けに実装されたものであり，ストリーム処理の各タスクは最大 16 個のノードに割り当てられる．アプリケーションは C 言語ベースで開発され，データ転送量も C 言語レベルのシミュレーションで解析された．*flee* による経路設定では，ソース-ディスティネーション・ペアに経路が設定される度に，その経路で利用されるチャネルのコストにデータ転送量が加算される．SoC 設計の初期段階におけるアルゴリズムレベルのシミュレーションでは，ネットワークの詳細設計が済んでおらず，パケット数やパケット長の情報は利用できない可能性がある．そこで，本評価ではデータ転送量の単位として，各ソース-ディスティネーション・ペアで発生する通信のビット長を用いた．

実装した JPEG コーデックのタスク割り当てを**図 6** に示す．16 ノードのうち，ノード 0-7 でデコード，ノード 8-15 でエンコード処理を行っている．JPEG コーデックのデータフローはシーケンシャルな構造を取っており，各タスクはパイプライン処理される．Viterbi デコーダのタスク割り当てを**図 7** に示す．Viterbi デコーダのデータフローでは，通信パターンに分散と収集を含む．IPsec アクセラレータのタスク割り当てを**図 8** に示す．これらのストリーム処理において，タスク間の交信パターンはほぼ直線状である．そのため，これらのアプリケーションを 2 次元メッシュとトーラスに対して近接交信量が最も大きくなるように手でマッピングした．タスクと対象ネットワークの相対位置により，他にも近接交信量が最大になるマッピング手法が存在するが，性能差はほとんどないと考えられる．

比較のため，実アプリケーションのトレースデータに加え，ユニフォームトラフィックも評価に用いる．このユニフォームトラフィックでは，全てのノードが自分以外のノードにランダムにパケットを送信する．パケット長は 259 フリットとし，そのうち 2 フリットはヘッダとした．

5.2 シミュレーション結果

5.2.1 2 次元メッシュおよびトーラス

3 種類のストリームアプリケーションとユニフォームトラフィックを用いて，Turn-Model ベースの *flee* ルーティング法と dimension-order ルーティングを比較する．

図 9 と**図 10** は，Viterbi トレースを用いた際のスループット（accepted traffic）とレイテンシを表しており，前者はメッシュ，後者はトーラスでの評価結果である．グラフ中の “Flee” は *flee*，“DOR” は dimension-order ルーティングを表し，それぞれの平均ホップ数を括弧内に示した．2 次元メッシュトポロジ（図 9）では，*flee* の平均ホップ数は 2.52，dimension-order ルーティングは 1.84 となり，*flee* が実際に非最短経路を取っていることが確認できる．このように非最短経路を導入することで，*flee* ルーティング法は経路を分散させ，dimension-order ルーティングよりスループットを 14.2%向上できた．一方，2 次元トーラストポロジ（図 10）では，*flee* は dimension-order ルーティングよりスループットを 22.2%向上させることができた．

図 11 と**図 12** は，それぞれメッシュとトーラスにおける JPEG トレースを用いた際のスループットとレイテンシのグラフである．この JPEG の実装では，メインストリームのデータ処理は逐次的に実行される．さらに，逐次的に実行される各タスクは，平均ホップ数が最小になるように配置されているため，ほとんどのノード間通信は隣接ノード間だけで完結している．そのため，*flee* で非最短経路はほとんど利用されず，dimension-order ルーティングとの性能差もほとんど生じなかった．

図 13 と**図 14** は，それぞれメッシュとトーラスにおける IPsec トレースを用いた際のスループットとレイテンシのグラフである．前者のグラフでは，*flee* は dimension-order ルーティングに対し 28.6%スループットを向上させた．全てのデータストリームは，各種暗号処理コアで暗号化や復号化，ハッシュ化されるため，暗号処理コアにトラフィックが集中しやすい．IPsec トレースでも非最短経路を導入することで Viterbi トレースと同様，トラフィックを分散させ，混雑を緩和することができた．

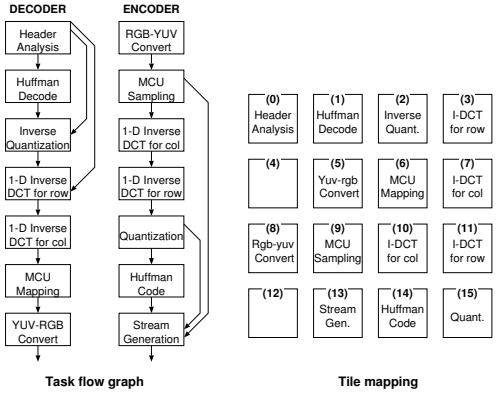

図 6 JPEG コーデックのタスクの流れとノードへの割り当て．ノード 0-7 でデコード，ノード 8-15 でエンコード処理を行う．

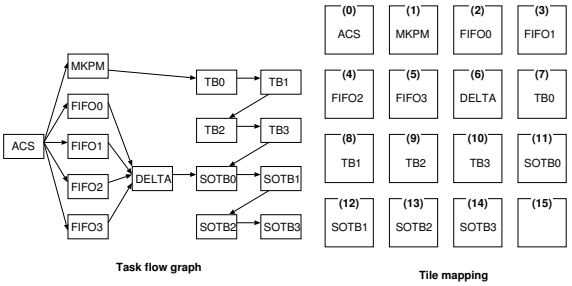

図 7 Viterbi デコーダのタスクの流れとノードへの割り当て．入力データは Add-Compare-Select（ACS）回路に転送される．ACS は Branch Metric を生成し FIFO に格納する．ACS はさらに Path Metric を生成する．Make Path Metric（MKPM）で最も確からしい状態が選ばれ，DELTA で最も確からしい Branch Metric が選ばれる．最後に，Trace Back（TB）と Soft Output Trace Back（SOTB）で復号結果を生成する．

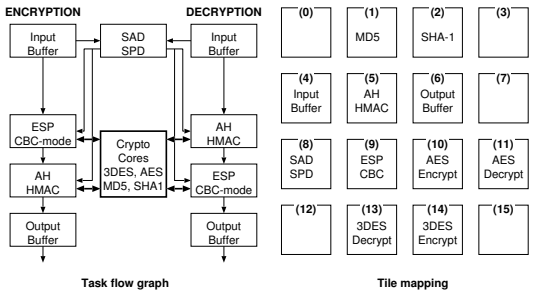

図 8 IPsec アクセラレータのタスクの流れとノードへの割り当て．入力パケットに対する IPsec 適用のルールは Security Policy Database（SPD）で管理し，暗号アルゴリズムや鍵などは Security Association Database（SAD）で保持する．Encapsulated Security Payload（ESP）では Triple-DES-CBC や AES-CBC を使って暗号化／復号化する．Authentication Header（AH）では HMAC-MD5 や HMAC-SHA-1 でハッシュ値を計算する．

本来，*flee* ルーティング法は高い局所性を持ったトラフィックパターンに対処するために設計されている．ここではワーストケースでの *flee* の振る舞いを調べるため，局所性のないユニフォームトラフィック上で評価を行った．**図 15** と **図 16** は，それぞれメッシュとトーラスにおけるユニフォームトラフィックを用いた際のスループットとレイテンシのグラフである．ユニフォームトラフィックでは dimension-order ルーティングによって十分に経路を分散させることができており，非最短経路を導入してもこれ以上の経路分

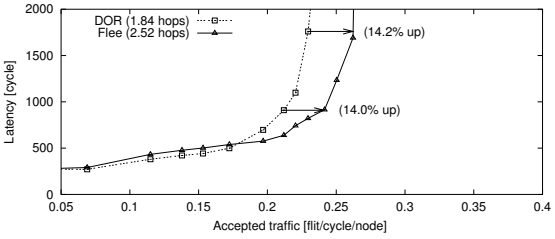

図 9 Viterbi トレース（4 × 4 メッシュ）．

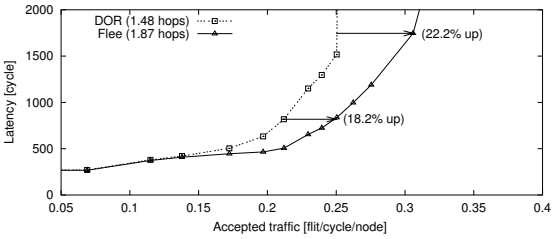

図 10 Viterbi トレース（4 × 4 トーラス）．

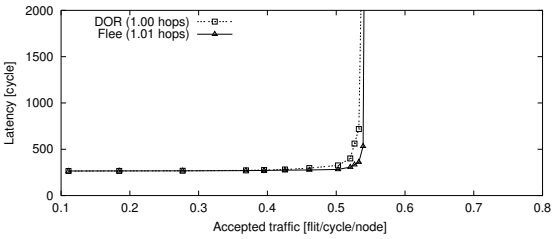

図 11 JPEG トレース（4 × 4 メッシュ）．

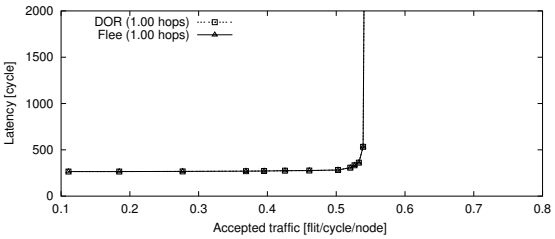

図 12 JPEG トレース（4 × 4 トーラス）．

散は望めない．そのため，非最短経路の導入はネットワークリソースを余計に消費するだけで hot-spot を緩和できず，結果のグラフでも *flee* の性能は dimension-order ルーティングに劣っている．グラフ中の平均ホップ数を見ると，*flee* ルーティング法はユニフォームトラフィックにおいてもいくつか非最短経路を取っていることがわかる．これは，analysis record に記された優先度順に，すでに設定した経路と重複しないように経路を張るからである．

ストリーム処理ではトラフィックに高い局所性を含むことが多い．このような通信パターンにおいて，*flee* ルーティング法は経路を分散させ，性能を向上させることができた．

5.2.2 リンク欠損あり 2 次元メッシュ

本節では，リンク欠損のあるメッシュ，つまり，利用可能なリンク数が少なくなった状態で *flee* ルーティング法の評価を行う．**図 17** はリンク欠損のないメッシュで Viterbi トレースを用いた際のスループットとレイテンシのグラフ，**図 18** はリンク欠損を 2 つ含むメッシュでのグラフである．リンク欠損なしのグラフ（図 17）では，*flee* の平均ホップ数は 3.01，他の最短経路のみを用いるアルゴリズムは 1.84 となった．*flee* は平均ホップ数が増えたにもかかわらず，他

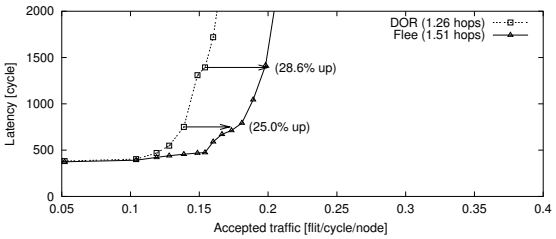

図 13 IPsec トレース（4 × 4 メッシュ）.

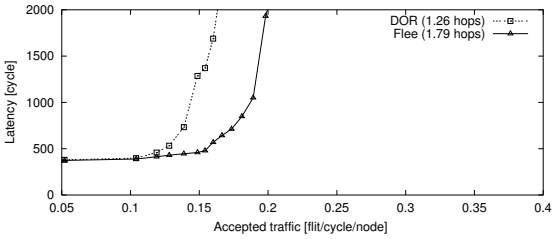

図 14 IPsec トレース（4 × 4 トーラス）.

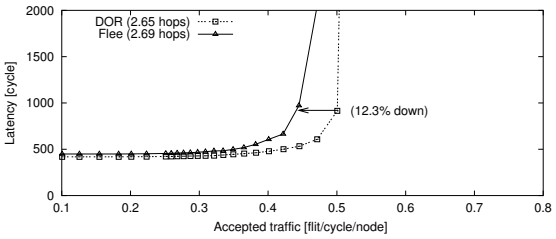

図 15 ユニフォームトラフィック（4 × 4 メッシュ）.

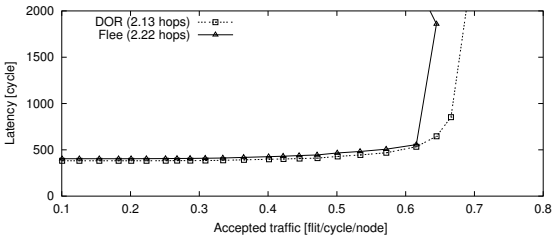

図 16 ユニフォームトラフィック（4 × 4 トーラス）.

のアルゴリズムより 12.0%程度スループットが向上している．一方，リンク欠損を含む評価結果（図 18）では，*flee* は他の経路選択アルゴリズムより性能が優れるものの，その優位性はかなり失われた．*flee* では，非最短経路の利用によって増えたホップ数の分だけネットワークリソースを多く消費するが，リンクの欠損によって代替経路の数が制限されると，非最短経路の利用に払うコストが無視できなくなる．そのため，リンク欠損時に非最短経路を取っても，リンク欠損なし時ほど性能が伸びない．

JPEG および IPsec トレースを用いた際のリンク欠損なしとリンク欠損時のスループット／レイテンシのグラフは，紙面の都合から割愛する．JPEG に関しては，5.2.1 節と同様の理由で，up*/down*ルールにおいても *flee* と他の経路選択アルゴリズムの性能に差異はほとんど出なかった．IPsec では，*flee* はリンク欠損なし時に他の経路選択アルゴリズムに対し優位性を持つものの，Viterbi と同様の理由で，リンクが欠損するとその優位性は失われた．

リンク欠損なし時の結果より，*flee* ルーティング法は up*/down*ルールにおいても，ストリーム処理におけるトラフィックの高い局所性を分散させ，スループットを向上できた．しかし，リンク欠損によって利用可能なリンク数が減ると，非最短経路の利用に払うコストが無視できなくなり，リンク欠損なし時ほど性能は向上しなかった．

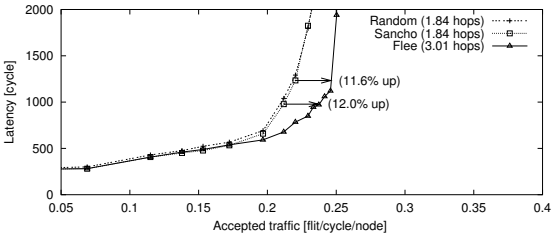

図 17 Viterbi トレース（リンク欠損なし 4 × 4 メッシュ）.

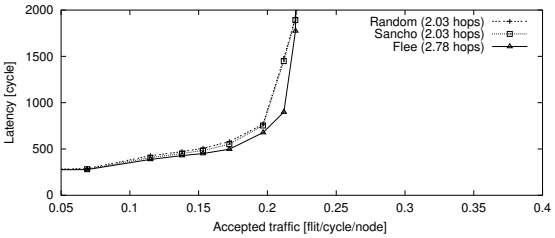

図 18 Viterbi トレース（2 リンク欠損あり 4 × 4 メッシュ）

5.2.3 静的解析の影響

flee ルーティング法における analysis record の影響を確認するため，完全な analysis record と，データ転送量が判明していない不完全な analysis record で性能比較を行った．両者の性能比較は 5.2.1 節と 5.2.2 節で示した全てのシミュレーション条件下で行ったが，ここでは Viterbi トレースでの性能比較を示す．

Turn-Model ルールを用いてメッシュで測定した結果を**図 19** に，トーラスでの結果を**図 20** に示す．また，up*/down*ルールを用いてリンク欠損なしメッシュで測定した結果を**図 21** に，2 リンク欠損ありメッシュでの結果を**図 22** に示す．グラフ中の “Flee.in” は不完全な analysis record を用いた場合，“Flee” は完全な analysis record を用いた場合を表している．不完全な analysis record を用いた *flee* の性能は，図 21 では完全な analysis record を用いた場合と変わらないが，図 19 では他の最短経路ルーティングと同程度の性能しか出ていない．このように不完全な analysis record を用いると *flee* の性能は安定しない．これは，各ソース-ディスティネーション・ペアのデータ転送量が経路を最適化する上で重要な要素であることを示している．4 章で述べたとおり，多くのストリーム処理アプリケーションでは設計時にデータ転送量を見積もることができる．しかしながら，データフローが動的に決まるなど，経路設定段階に及んでもデータ転送量が見積もることができない可能性もあり，その場合に用いられる不完全な analysis record では，完全な analysis record ほど安定して性能を向上できないことがわかった．

6. ま と め

本論文では，チップ内ネットワーク向けに *flee* と呼ばれるデッドロックフリーな固定型ルーティング法を提案した．チップ内ネットワークでは SoC の設計段階で通信パターン

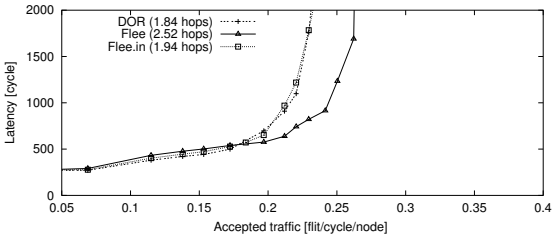

図 19 不完全な analysis record での Viterbi トレース（4 × 4 メッシュ，Turn-Model ルール）.

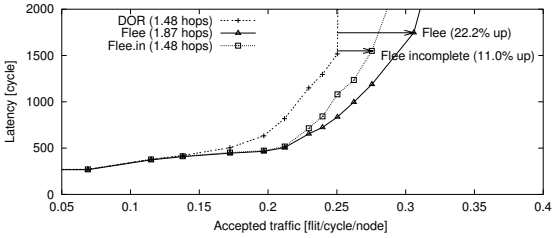

図 20 不完全な analysis record での Viterbi トレース（4 × 4 トーラス，Turn-Model ルール）.

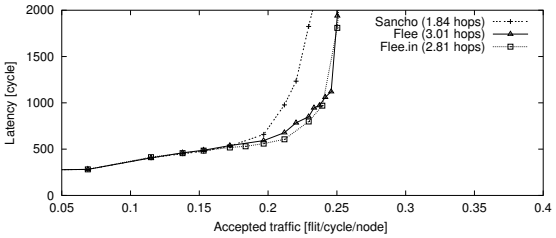

図 21 不完全な analysis record での Viterbi トレース（リンク欠損なし 4 × 4 メッシュ，up*/down*ルール）.

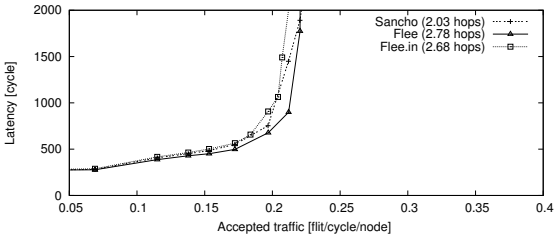

図 22 不完全な analysis record での Viterbi トレース（2 リンク欠損あり 4 × 4 メッシュ，up*/down*ルール）.

を予測できる．これらの通信パターンには強い局所性が出やすいが，最短経路だけでは経路長が短く十分にトラフィックを分散できるとは限らない．そこで，*flee* ルーティング法では，非最短経路を導入することで代替経路の候補数を増やし，高負荷なチャネルを確実に回避する．*flee* では SoC 設計の初期段階から得られるデータ転送量の見積もりをもとに，多量のデータが流れるソース-ディスティネーション・ペアに優先的に最短経路を割り当てる．一方，データ転送量の少ないペアは高負荷なチャネルを避けるように経路が張られるため，非最短経路を取ることがある．実際のストリームアプリケーションを用いたシミュレーションでは，*flee* ルーティング法は非最短経路を用いることで経路を分散でき，既存の最短経路ルーティング法より最大 28.6%スループットを向上できた．*flee* ルーティング法はトラフィックパターンが完全に解析できた場合に加え，データ転送量が不明な場合にも適用できる．両者の性能比較より，データ転送量を考慮した analysis record を利用することができれば安定してスループットを向上できることがわかった．

参 考 文 献

1) Taylor, M. B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffmann, H., Johnson, P., Lee, J.-W., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S. and Agarwal, A.: The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs, *IEEE Micro*, Vol. 22, No. 2, pp. 25–35 (2002).
2) picoChip Designs Ltd.: picoArray (2005). http://www.picochip.com/technology/picoarray/.
3) Dally, W. J. and Towles, B.: Route Packets, Not Wires: On-Chip Interconnection Networks, *Proceedings of the 38th Design Automation Conference*, pp. 684–689 (2001).
4) Liang, J., Laffely, A., Srinivasan, S. and Tessier, R.: An Architecture and Compiler for Scalable On-Chip Communication, *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 12, No. 7, pp. 711–726 (2004).
5) Marescaux, T., Bartic, A., Verkest, D., Vernalde, S. and Lauwereins, R.: Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs, *Proceedings of the Field-Programmable Logic and Applications (FPL)*, pp. 795–805 (2002).
6) Hu, J. and Marculescu, R.: Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints, *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 233–239 (2003).
7) Banerjee, N., Vellanki, P. and Chatha, K. S.: A Power and Performance Model for Network-on-Chip Architectures, *Proceedings of Design Automation and Test in Europe Conference (DATE 2004)*, pp.1250–1255 (2004).
8) Anjo, K., Yamada, Y., Koibuchi, M., Jouraku, A. and Amano, H.: BLACK-BUS: A New Data-Transfer Technique using Local Address on Networks-on-Chips, *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, p. 10a (2004).
9) Owens, J. D., Kapasi, U. J., Mattson, P., Towles, B., Serebrin, B., Rixner, S. and Dally, W. J.: Media Processing Applications on the Imagine Stream Processor, *Proceedings of the IEEE International Conference on Computer Design*, pp. 295–302 (2002).
10) Ho, W. H. and Pinkston, T. M.: A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns, *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture*, pp. 377–388 (2003).
11) Dally, W. J. and Seitz, C. L.: Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, *IEEE Transaction on Computers*, Vol. 36, No. 5, pp. 547–553 (1987).
12) Sancho, J. C. and Robles, A.: Improving the Up*/Down* Routing Scheme for Networks of Workstations, *Proceedings of the European Conference on Parallel Computing*, pp. 882–889 (2000).
13) Koibuchi, M., Jouraku, A. and Amano, H.: Path Selection Algorithm: The Strategy for Designing Deterministic Routing from Alternative Paths, *PARALLEL COMPUTING*, Vol. 31, No. 1, pp. 117–130 (2005).
14) Glass, C. J. and Ni, L. M.: The Turn Model for Adaptive Routing, *Proceedings of International Symposium on Computer Architecture*, pp. 278–287 (1992).
15) Chakraborty, K. and Mazumder, P.: *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*, Prentice Hall PTR (2002).
16) NEC エレクトロニクス: 動的再構成プロセッサ Dynamically Reconfigurable Processor (DRP) (2005). http://www.necel.com/ja/techhighlights/drp/.